UNITED STATES PATENT APPLICATION

LATENCY TOLERANT DISTRIBUTED SHARED MEMORY MULTIPROCESSOR COMPUTER

INVENTORS

Steven L. Scott

Gregory J. Faanes

Brick Stephenson

William T. Moore, Jr.

Mark Birrittella

James Schwarzmeier

Peter Klausler

David Resnick

Steve Oberlin

Rabin Sugumar

Schwegman, Lundberg, Woessner, & Kluth, P.A.
1600 TCF Tower
121 South Eighth Street
Minneapolis, Minnesota 55402
ATTORNEY DOCKET 1376.700US1

LATENCY TOLERANT DISTRIBUTED SHARED MEMORY MULTIPROCESSOR COMPUTER

Related Applications

This application is related to U.S. Patent Application No. _____, entitled "Multistream Processing System and Method", filed on even date herewith; to U.S. Patent Application No. _____, entitled "System and Method for Synchronizing Memory Transfers", Serial No. _____, filed on even date herewith; to U.S. Patent Application No. _____, entitled "Decoupled Store Address and Data in a Multiprocessor System", filed on even date herewith; to U.S. Patent Application No. _____, entitled "Decoupled Vector Architecture", filed on even date herewith; to U.S. Patent Application No. _____, entitled "Relaxed Memory Consistency Model", filed on even date herewith; to U.S. Patent Application No. _____, entitled "Remote Translation Mechanism for a Multinode System", filed on even date herewith; and to U.S. Patent Application No. _____, entitled "Method and Apparatus for Local Synchronizations in a Vector Processor System", filed on even date herewith, each of which is incorporated herein by reference.

Technical Field

This document relates to computer system technology, and, in particular, to a computer system tolerant of memory access latency.

Background

Distributed computer system designs based on clusters of relatively inexpensive microprocessors have become popular. However, there is still a need for Vector Processing Computer Systems that are able to handle calculation-intensive problems on a large amount of data. Traditional vector systems do not scale to a large number of processors due to their system architectures. Previous vector machines tended to have a limited number of processors clustered around a shared memory. The shared memory was developed to minimize communication costs when sharing data between processors.

Microprocessor-based machines on the other hand, suffer from limitations in the number of outstanding memory references they can handle. This makes it difficult for microprocessor-based machines to tolerate high memory access latencies. In addition, microprocessor-based machines use a memory subsystem based on cache line granularity, which is inefficient when accessing single words. What is needed is a computer system structure scalable to a large number of processors yet can tolerate hundreds of outstanding memory references.

Brief Description of the Drawings

Figures 1- 4 show block diagrams of MSPs scaled from 2 processors, 2 cache memories, and 2 local memory ports in Figure 1, to 4 processors, 4 cache memories, and 16 local memory ports in Figure 4.

Figures 5 - 8 show block diagrams processing nodes scaled from 2 MSPs and 2 local memories in Figure 5, to 4 MSPs and 16 local memories in Figure 8.

Figure 9 shows a block diagram of one embodiment of a processor for use in a computer system.

Figure 10 shows a block diagram of one embodiment of a local memory used in a computer system.

Detailed Description

In the following detailed description, reference is made to the accompanying drawings which form a part thereof, and in which is shown by way of illustration specific embodiments in which the invention may be practiced. Other embodiments may be used and structural changes may be made without departing from the scope of the present invention.

The computer system structure of the present application is comprised of interconnected processing nodes. Each processing node is comprised of a number of Multi-Streaming Processors (MSPs), a number of cache memories, and a number of local memories.

Figure 1shows one embodiment of an MSP 100. The MSP 100 includes two processors 110 and two cache memories 120.

In some embodiments, an MSP includes synchronization features that allow for low-latency synchronization. In various embodiments, these features allow for synchronization of multiple processors within an MSP and among various MSPs. This allows individual processors to be applied at different levels of parallelism by a compiler, making the processors more flexible than a single processor. For a fuller description of these synchronization features, please refer to the U.S. patent applications entitled "Multistream Processing System and Method", filed on even date herewith, "System and Method for Synchronizing Memory Transfers", file on even date herewith, and "Method and Apparatus for Local Synchronizations in a Vector Precessing System", filed on even date herewith, the descriptions of which are hereby incorporated by reference.

Figure 9 shows one embodiment of processor 110. Each processor 110 is composed of a scalar processor 910 and two vector pipes 930. The scalar and vector unit are decoupled with respect to instruction execution and memory accesses. Decoupling with respect to instruction execution means the scalar unit can run ahead of the vector unit to resolve control flow issues and execute address arithmetic. Decoupling with respect to memory accesses means both scalar and vector loads are issued as soon as possible after instruction dispatch. Instructions that depend upon load values are dispatched to queues where they await the arrival of the load data. Store addresses are computed early and their addresses saved for later use. Each scalar processor 910 is capable of decoding and dispatching one vector instruction (and accompanying scalar operand) per cycle. Instructions are sent in order to the vector units, and any necessary scalar operands are sent later after the vector instructions have flowed through the scalar unit's integer or floating point pipeline and read the specified registers. Vector instructions are not sent speculatively; that is, the flow control and any previous trap conditions are resolved before sending the instructions to the vector unit. For a further description of decoupled vector architecture please refer to the U.S. patent application entitled "Decoupled Vector Architecture", filed on even date herewith, the description of which is hereby incorporated by reference.

In another embodiment, processor 110 contains a cache memory 920 for scalar references only. Local MSP cache coherence is maintained by requiring all data in processor cache memory 920 to be contained in MSP cache memory 120.

Within MSP 100 in Figure 1, each cache memory 120 is shared by each processor 110. Each cache memory includes two processor ports 140 to allow sharing by processors 900, and one memory port 130 for accessing local memory 1000. Thus each MSP 100 contains two local memory ports 130.

Figure 2 shows another embodiment of an MSP 200. The MSP 200 is composed of two processors 900 and two cache memories 120. Each cache memory 120 includes two processor ports 140 to allow sharing by each processor 110 and also includes two memory ports 130 for addressing local memory 1000. Thus each MSP 200 contains four local memory ports 130.

Figure 3 shows another embodiment of an MSP 300. The MSP 300 is composed of two processors 900 and two cache memories 120. Each cache memory includes four processor ports 140 and also includes four memory ports 130 for addressing local memory 1000. Thus in this embodiment each MSP 300 contains eight local memory ports 130.

Figure 4 shows another embodiment of an MSP 400. The MSP 400 is composed of four processors 900 and four cache memories 120. Each cache memory 120 includes four processor ports 140 to allow sharing by the four processors 900. The processor connections 410 are connected round robin across the cache memory ports 140. Each cache memory 120 also includes four ports 130 for addressing local memory 1000. Thus in this embodiment each MSP 400 contains sixteen local memory ports 130.

Figure 5 shows one embodiment of a processing node 500. The processing node 500 includes two MSPs 100 each having two local memory ports 130, one I/O channel controller 510, and two local memories 1000. Each local memory includes two MSP ports 1010. Thus each processor 110 in Figure 1 can access each local memory 1000 in Figure 5.

Figure 6 shows another embodiment of a processing node 600. The processing node 600 includes four MSPs 200 each having four local memory ports 130, one I/O channel controller 510, and four local memories 1000. Each local memory includes four MSP ports 1010. Thus each processor 110 in Figure 2 has access to each local memory 1000 in Figure 6.

Figure 7 shows another embodiment of a processing node 700. The processing node 700 includes two MSPs 300 each having eight local memory ports 130, one I/O channel

controller 510, and eight local memories 1000. Each local memory includes two MSP ports 1010. Thus each processor 110 in Figure 3 has access to each local memory 1000 in Figure 7.

Figure 8 shows another embodiment of a processing node 800. The processing node 800 includes four MSPs 400 each having sixteen local memory ports 130, two I/O channel controllers 510, and sixteen local memories 1000. Each local memory includes four MSP ports 1010. Thus each processor 110 in Figure 4 can access each local memory 1000 in Figure 8.

The embodiments illustrated in figures 5 through 8 show how the computer system can be scaled from two to sixteen processors. The embodiments allow further scaling when the processing nodes 500, 600, 700, 800 are interconnected. Each local memory 1000 of each processing node 500, 600, 700, 800 includes two network ports 1030 for interconnecting the local memories 1000.

In one embodiment, processing nodes 500, 600, 700, 800 are interconnected by connecting the local memories 1000 of one processing node 500, 600, 700, 800 to the corresponding local memory 1000 of two other processing nodes. In this way four processing nodes can be interconnected into a two-dimensional hypercube, or square. Each local memory 1000 of each processing node 500, 600, 700, 800 resides in one independent, parallel slice of the computer system and connectivity is only provided between corresponding local memories 1000 on the processing nodes. Thus for the processing node 800 of Figure 8 there are sixteen parallel, independent networks.

The region of memory belonging to each independent network slice at a given node is called a section. In some embodiments, cache lines are mapped round-robin across the 16 sections of a node, using physical address bits 8..5. Thus, the memory on a single node is uniformly shared by all MSPs on that node via the MSP_to_M-chip, on-node network. An M-chip functions as the routing hub for all traffic to memory, I/O, and to the network for all data from a node for a single slice of the address space. The M chip is one sector of a slice and supports 1 or 2 daughter cards or 4 or 8 memory-channels. In some embodiments, each network slice is interconnected via its own independent network, which connects together all

M-chips as shown in FIG. 8 (i.e., M0 - M15) in system belonging to the same slice. Each M-chip M0 - M15 contains two network ports 1030 for this purpose. Memory references by an MSP are first routed to the local M-chip of the appropriate slice. From there, they either access local memory, or route to the correct destination node on the network for that slice. Each slice of the machine independently handles all memory accesses and routing for addresses that map to that slice.

In another embodiment, one network port 1030 of the local memory 1000 of a processing node 500, 600, 700, 800 is connected to the corresponding local memory 1000 of the neighboring board and the other network port 1030 is connected to a router. Thus, for two processing nodes 800 of the embodiment shown in Figure 8, there are thirty-two local memory-to-router connections and these connect to thirty-two parallel, independent networks.

In another embodiment, an eight-ported router is used to connect four local memories 1000 leaving four router ports available for router-to-router connections. Each of the thirty-two parallel networks grows as a hypercube up to sixteen routers per parallel slice (512 total MSPs) and then as a 2D torus up to a maximum of a 4096 MSP system.

Thus the computer system supports scaling from a one MSP system of two processors to a 4096 MSP system of 4 processors per MSP.

Figure 10 shows one embodiment of local memory 1000 used in the processing node 500 of Figure 5. In this embodiment, local memory includes two MSP ports 1010, two Cache Coherence Directories 1040, a crossbar switch 1020, two network ports 1030, a Remote Address Translation Table (RTT) 1050, and RAM 1060. Remote Translation Table (RTT) 1050 translates addresses originating at remote processing nodes 500, 600, 700, 800 to physical addresses at the local node. In some embodiments, this includes providing a virtual memory address at a source node, determining that the virtual memory address is to be sent to a remote node, sending the virtual memory address to the remote node, and translating the virtual memory address on the remote node into a physical memory address using a RTT. The RTT contains translation information for an entire virtual memory address space associated with the remote node. Another embodiment of RTT provides for translating a virtual memory address in a multi-node system. The method includes providing

a virtual memory address on a local node by using a virtual address of a load or a store instruction, identifying a virtual node associated with the virtual memory address, and determining if the virtual node corresponds to the local node. If the virtual node corresponds to the local node, then the method includes translating the virtual memory address into a local physical memory address on the local node. If, instead, the virtual node corresponds to a remote node, then the method includes sending the virtual memory address to the remote node, and translating the virtual memory address into a physical memory address on the remote node. For a further description of RTTs please refer to the U.S. patent application entitled "Remote Translation Mechanism for a Multi-node System", U.S. Application No. 10/235,898, filed September 4, 2002; "Remote Translation Mechanism for a Multinode System", filed on even date herewith, and "Method for Sharing a Memory within an Application Using Scalable Hardware Resources", filed on even date herewith, the descriptions of which are hereby incorporated by reference.

The Cache Coherence Directories 1040 keep track of the contents of the MSP cache memories 120 in the processing node 500. Only cache memories 120 from the local processing node 500, 600, 700, 800 are allowed to cache data from that node. Each coherence directory 1040 included in the local memory 1000 corresponds to one bank of MSP cache memory 120. There is one cache coherence directory 1040 entry for every MSP cache line on the processing node 500, 600, 700, 800. Thus the directories hold entries only for a subset of the total local memory lines, and do not need to include entries for all of the non-cached memory lines. The Cache Coherence Directories 1040 are designed to support very high access rates generated by the processing nodes 500. For a further description of Cache Coherence Directories please refer to the application entitled "Optimized High Bandwidth Cache Coherence Mechanism", U.S. Application No. 10/368,090, filed February 18, 2003, the description of which is hereby incorporated by reference.

In addition to promoting scaling, the partitioning in the above embodiments results in a scalable, shared address space distributed across the entire computer system that is tolerant of memory latencies by allowing direct load store access to all memory. This improves sustained bandwidth capabilities in the system. All activity (E.g. cache, local memory, or network transfers) relating to an area of memory stays within the corresponding system slice.

Most single processor tasks will run local to the node. Latency of a memory request to a remote processing node depends upon the distance to the remote processing node and the level of contention in the network. For a further description of load store operations please refer to the U.S. patent applications entitled "Indirectly Addressed Vector Load-Operate-Store Method and Apparatus", filed on even date herewith, "Method and Apparatus for Indirectly Addressed Vector Load-Add-Store Across Multi-processors", filed on even date herewith, and "System and Method for Processing Memory Instructions", filed on even date herewith, the descriptions of which are hereby incorporated by reference.